Grundkurs Turbo Pascal

von Henning Schwanbeck

Technische Universität Ilmenau
Dezember 2002

Inhaltsverzeichnis

1 Belegung der Funktionstasten

1.1 Allgemein

ESC	Befehl abbrechen/Fenster schließen
Alt F4	Fenster schließen

1.2 Dateiarbeit

F2	Datei speichern
F3	Datei öffnen
F9	Datei compilieren
Strg F9	Programm/Datei compilieren und ausführen
Alt F5	Umschaltung Ausgabebildschirm und zurück
Enter	Ausgabebildschirm schließen
Alt X	Turbo Pascal beenden

1.3 Editorbefehle (Auswahl)

EINF	Umschaltung Überschreibmodus/ Einfügemodus
ENTFf	Zeichen unter Cursor löschen
Backspace	Zeichen links vom Cursor löschen
Strg+Y	Löschen der aktuellen Zeile
Strg+N	Einfügen einer Zeile
Shift+ENTF	Ausschneiden von markiertem Text
Strg+EINF	Kopieren von markiertem Text
Shift+EINF	Einfügen von markiertem Text aus der Zwischenablage
Strg+ENTF	Löschen von markiertem Text
Strg+K+U	Markierten Text um ein Zeichen nach links rücken
Strg+K+I	Markierten Text um ein Zeichen nach rechts rücken

1.4 Hilfe

F1	Allgemeines Hilfemenü
Shift F1	Index der Hilfeebenen
Strg F1	Syntax des ausgewählten Befehls
Alt F1	Letzter Hilfetext

1.5 Hilfe im Internet

- http://www.rz/tu-ilmenau.de/∼schwan/tp
- www.google.com und dann das Stichwort *pascal* eingeben

- Newsgroups de.comp.pascal und viele andere !!!

2 Der Programmaufbau

2.1 Einleitung

Ein Turbo Pascal Programm besteht in seiner prinzipiellen Struktur aus einem Deklarationsteil und einem Anweisungsteil.

2.2 Der Deklarationsteil

Im Deklarationsteil werden die die im Anweisungsteil verwendeten Variablen deklariert. Dieser Abschnitt im Deklarationsteil beginnt mit dem reservierten Bezeichner *var*.

```
var i           : integer;
    s,yy        : string;
    fuchs, gans : real;
    a1,a2       : array[1..100,1..50] of real;
```

Auf die in diesem Programmbeispiel zurückgegriffenen Variablentypen wird im Laufe der nächsten Abschnitte eingegangen.

Ebenso können im Deklarationsteil folgende Parameter deklariert werden:

uses crt,graph; Einbinden von externen Units

label marke1, hier; Deklaration von Marken

const ee = 2.71e-5; Deklaration von Konstanten

type ganz = integer; Deklaration von neuen Variablentypen

procedure rechteck; Deklaration eines Unterprogramms namens *rechteck*

function kreis; Deklaration der Funktion namens *kreis*

2.3 Der Anweisungsteil

Im Anweisungsteil werden die auszuführenden Kommandos aufgeführt:

```
begin
  i:=3;
  i=i*i;
  fuchs:=0.0456;
  fuchs:=sqrt(fuchs);
end.
```

Der Anweisungsteil im Hauptprogramm ist zwischen den beiden Bezeichern *begin* und *end* eingeschlossen. Im Hauptprogramm ist der Punkt hinter dem *end*- Kommando sehr wichtig und darf nicht vergessen werden!

2.4 Hello World

Nachfolgend ist ein einfache Pascalprogramm dargestellt, welches eine Ausschrift auf dem Bildschirm ausgibt:

```
program Hello_World;
begin       { Hier beginnt der Anweisungsteil }
  writeln('Hello World');
end.       { Hier endet der Anweisungsteil }
           { und das Programm  }
```

2.5 Noch ein Programm

```
program Kreis;
  var radius, umfang, flaeche : real;
begin
 write('Geben Sie bitte den Kreisradius ein : '):
 readln(radius)

 {* jetzt wird der Umfang berechnet *}
 umfang:=2*pi*radius;
 writeln('Umfang= ',umfang);

 {* jetzt wird die Flaeche berechnet *}
 flaeche:=pi*radius*radius;  {* pi*spr(radius) *}
 writeln('Flaeche= ',flaeche);

 readln;  {* Wozu ist das gut??? *}
end.
```

2.6 Die Ein- und Ausgabe: readln - writeln

In den ersten Übungen steht die Eingabe von Variablenwerten und die Ausgabe von Berechnungsergebnissen im Vordergrund.

2.6.1 Die Ausgabe mit writeln

Mit dem *write-* oder *writeln-* Kommando können Inhalte von Variablen auf dem Bildschirm ausgegeben werden:

```
program Kreis;
  var radius : real;
begin
 radius:=0.03;
 writeln(radius);
```

```
 write(radius); { Was ist anders als bei writeln? }
end.
```

Es können auch Zeichenfolgen auf dem Bildschirm ausgegeben werden:

```
program ausgabe;
begin
 writeln('Heute ist ein sonniger Tag');
end.
```

Die auszugeben Zeichen oder Wörter werden in sogenannten *Hochkommas* eingeschlossen. Ebenso können Zeichen, Wörter und Variableninhalte in zusammenhängender Form auf dem Bildschirm ausgegeben werden.

```
program ausgabe;
  var radius : real;
begin
 radius:=0.03;
 writeln('Der Radius betraegt: ',radius);
end.
```

Die Trennung der auszugebenden Werte wird mit einem Komma vorgenommen.

2.6.2 Die Eingabe mit readln

Mit dem *readln*- Kommando können Inhalte von Variablen von der Tastatur eingelesen werden. Im nachfolgenden Programmbeispiel wird der Wert der Variablen namens *radius* von der Tastatur eigelesen:

```
program Kreis;
  var radius : real;
begin
 readln(radius);
 writeln('Der Radius betraegt: ',radius);
end.
```

2.6.3 Die Ausgabeformatierung

Die Darstellung und Teilung von Gleitkommazahlen in Mantisse und Exponent ist nicht sehr angenehm. Daher kann mit dem Kommando *writeln* ein Formatierung der Ausgabe vorgenommen werden:

```
program Kreis;
  var radius : real;
begin
 radius:=0.03;
 writeln('Der Radius betraegt: ',radius);
```

```
 writeln('Der Radius betraegt: ',radius:12);
 writeln('Der Radius betraegt: ',radius:12:4); { rechtsbuendig }
 writeln('Der Radius betraegt: ',radius:0:4);  { linksbuendig }
 readln;  { Was bewirkt das readln an dieser Stelle ? => Druecke Enter}
end.
```

Probieren Sie die verschiedenen Varianten der Ausgabeformatierung aus und sehen Sie sich den Effekt an! Die erste Zahl hinter dem Doppelpunkt *:12* gibt die Breite des Ausgabefeldes an. Die zweite Zahl *4* gibt die Anzahl der Nachkommastellen an. Eine Formatierung in der Form *:0:4* ergibt eine linksbuendige Ausgabe des Variableninhalts.

2.7 Übungsaufgaben

1. Berechnen Sie die mittlere Geschwindigkeit eines Fahrzeuges (Input: Weg und Zeit; Output: Geschwindigkeit)!

2. Berechnen Sie unter Verwendung des Pythagoras-Satzes die Diagonale eines Rechtecks (Input: Seitenlänge a und b; Output: Umfang, Fläche und Diagonale)!

3. Berechnen Sie das Volumen einer Kugel (Input: Kugelradius; Output: Kugelvolumen)! (Kugelvolumen: $V_K = \frac{4}{3} \cdot \pi \cdot r^3$)

4. Berechnen Sie die kinetische Energie eines Fahrzeuges mit der Geschwindigkeit v (Input: v, Output: kinetische Energie E_{kin})

5. Berechnen Sie die Gewichtskraft als Sonderfall der Gravitationskraft $F_G = m \cdot g$ (Input: Masse m, Output: Kraft F_G)!

6. Üben Sie an den vorangegangen Aufgaben die Ausgabeformatierung mit dem *writeln* Kommando!

3 Bedingungsanweisungen

3.1 Einleitung

Die *if-then-else*-Anweisung wertet eine Bedingung aus. Wenn die Bedingung wahr ist dann wird der *then*-Zweig ausgeführt -ansonsten werden die im *else*-Zweig aufgeführten Anweisungen ausgeführt. Der *else*-Zweig ist optional.

3.2 Programmbeispiele

3.2.1 if-then-else

```
if x<0 then begin
              writeln('Kann die Wurzel nicht berechnen !')
            end
       else begin
              y:=sqrt(x);
              writeln(y);
            end;
```

3.2.2 if-then

```
if x>0 then begin
              y:=sqrt(x);
              writeln(y);
            end;
```

3.2.3 Verschachtelte if-then-else-Anweisung

```
if x<0 then begin
              writeln('Kann die Wurzel nicht berechnen !')
            end
       else if x>100 then begin
                            y:=x*x;
                            writeln(y);
                          end;
            end;
```

3.2.4 Die Fallunterscheidung mittels der case- Anweisung

Die bedingte *if-then-else*- Anweisung ist sehr ungünstig, wenn aus mehreren Alternativen ausgewählt werden soll. Das ist zum Beispiel immer der Fall, wenn man eine Auswahl aus einem Menü treffen soll.

Nachfolgend ein Programmauszug zu der case- Anweisung:

```
write('Bitte ein Zeichen eingeben: '); readln(ch);
write('Das ist ein');
ch:=upcase(ch);

case ch of  'A','E','I','O','U': writeln(' Vokal.');
            'A'..'Z'           : writeln(' Konsonant.');
            '0'..'9'           : writeln('e Ziffer.');
     else write(' Sonderzeichen.');  { else Zweig ist optional! }
end;
```

3.3 Übungsaufgaben

1. Berechnen Sie unter Verwendung des Pythagoras-Satzes die Diagonale eines Rechtecks (Input: Seitenlänge a und b; Output: Umfang, Fläche und Diagonale)! Testen Sie die eingelesenen Seitenlängen unter Verwendung der *if-then-else-* Anweisung auf positive oder negative Werte und lösen Sie entsprechende Aktionen aus (z. B. Fehlermeldung)!

2. Bestimmen Sie von drei einzugebenen ganzen Zahlen mit Hilfe der *if-then-else-*Anweisung die größte Zahl!

3. Sortieren Sie drei ganze ganze Zahlen in aufsteigender Folge der Größe nach unter Verwendung der *if-then-else-* Anweisung!

4 Wiederholungsanweisungen

4.1 Die for-to-do- Anweisung

Bei einer *for-to-do*-Anweisung ist die Anzahl der Wiederholungen bekannt und kann während der Ausführung nicht geändert werden. Die Anzahl der Weiderholungen ergibt sich aus der Differenz von zwischen Anfangswert und Endwert. Die Laufvariable, der Anfangswert und der Endwert müssen vom gleichen ordinalen Variablentyp sein. Die Schrittweite beträgt immer 1 (*for-to-do*) oder -1 (*for-downto-do*). Der Wert der Laufvariablen ändert sich damit bei jedem Durchlaufen der Schleife um den Wert 1 bzw -1.

Nachfolgende Programmbeispiel gibt bei jedem Schleifendurchlauf den Wert der Laufvariablen `i` aus:

```
program schleife_1;
  var i : integer;
begin
  for i:=1 to 11 do    { Startwert des Index = 1 }
  begin                { wenn i=11 dann Schluss }
    writeln('i=',i);
  end;
  readln;
end.
```

Damit wird die Veränderung der Laufvariable - auch Schleifenindex genannt - ersichtlich. Im zweiten Programmbespiel wird mit einer `for-to-do`- Anweisung die Summe von 99 Zufallszahlen berechnet:

```
program schleife_summe;
var i : integer;
    zz: real;
begin
  summe:=0;
  for i:=10 to 109 do   { Startwert des Index = 10 ! }
  begin                 { wenn i=110 dann Schluss    }
    zz:=random;         { Schrittweite ist immer 1 ! }
    writeln('i=',i,' ZZ= ',zz);
    summe:=summe+zz;
  end;
  writeln('Summe= ',summe:8:6);
end.
```

Der Anfangs- und Endwert einer Wiederholungsanweisung ist frei bestimmbar - entscheidend ist immer das relative Verhältnis dieser beiden Parameter.

4.2 Die repeat-until- Anweisung (Der Optimist!)

Bei einer `repeat-until`- Anweisung wird *zuerst* der Anweisungsteil ausgeführt und erst dann wird die Überprüfung des Abbruchskriteriums vorgenommen. Alle Parameter (a) Anfangswert, b) Endwert oder Abbruchbedingung und c) Schrittweite müssen bei einer `repeat-until`- Anweisung direkt vorgegeben werden!
Nachfolgendes Programmbeispiel berechnet die Summe von 117 Zufallszahlen:

```
var i : integer;
    zz: real;
begin
  summe:=0;
  i:=0; { Startwert des Index = 1 }
  repeat
    i:=i+1;                      { Schrittweite = 1 }
    zz:=random;
    writeln('i=',i,' ZZ= ',zz);  { optionale Ausgabe }
    summe:=summe+zz;
  until i=117;                   { wenn i=17 dann Schluss }
  writeln('Summe= ',summe:8:6);
end.
```

Aus diesem Programmbeispiel wird nochmals ersichtlich, daß der Anweisungsblock einer `repeat-until`- Anweisung immer mindestens einmal durchgeführt wird - auch wenn die Abbruchbedingung schon vor Beginn des Anweisungsblockes falsch ist!

4.3 Die while-do- Anweisung (Der Pessimist!)

Bei einer `while-do`- Anweisung wird zuerst das Abbruchkriterium geprüft. Ist das Ergebnis des Abbruchkriteriums wahr (true), dann wird der Anweisungsteil ausgeführt. Führt die Auswertung des Abbruchkriteriums zum Wert nein (false) dann wird die `while-do`-Anweisung beendet. In nachfolgendem Programmbeispiel wird nochmals die Summe von 100 Zufallszahlen unter Verwendung einer `while-do`- Anweisung ermittelt:

```
var i : integer;
    zz: real;
begin
  summe:=0;
  i:=0; { Startwert des Index = 1 }
  while i<100 do  { wenn i=100 dann Schluss }
  begin
    i:=i+1; { Schrittweite = 1}
    zz:=random;    writeln('i=',i,' ZZ= ',zz);
    summe:=summe+zz;
  end;
```

```
  writeln('Summe= ',summe:8:6);
end.
```

4.4 Übungsaufgaben

1. Geben Sie auf dem Bildschirm eine Zeile bestehend aus dem Zeichen = mit der

   a) *for-to-do*-Anweisung

   b) *repeat-until*-Anweisung

   c) *while-do*-Anweisung

   aus! Eine Bildschirmzeile hat eine eine Zeilenlänge von 80 Zeichen.

2. Geben Sie unter Verwendung der *while-do*-Anweisung die Fläche eines Rechteckes in Abhängigkeit von der Seitenlänge a im Bereich von 137 mm bis 153 mm in Schritten von 2 mm auf dem Bildschirm aus! Die Länge der Seite b ist konstant und beträgt 5 cm.

3. Berechnen Sie die Summe von zehn einzugebenden Zahlen mit der *for-to-do*-Anweisung (Input: 10 Zahlen; Output: Summe)!

4. Berechnen Sie die Summe von 100 Zufallszahlen mit der *for-to-do*-Anweisung und der *random- Funktion* (Output: Summe)!

5. Berechnen Sie die Summe sowie das Maximum und das Minimum von n Zufallszahlen mit der *for-to-do*-Anweisung und der *random- Funktion* (Input: n; Output: Summe, Minimum, Maximum)!

6. Berechnen Sie die Fakultät einer Zahl f mit Hilfe der

   a) *for-to-do*-Anweisung

   b) *repeat-until*-Anweisung

   c) *while-do*-Anweisung

7. Lesen Sie unter Verwendung der *repeat-until*-Anweisung positve Zahlen ein und bestimmen Sie die Summe dieser Zahlen! Bei Eingabe einer negativen Zahl soll die Berechnung abgebrochen werden.

8. Geben Sie das Alphabet von A..Z unter Verwendung der *repeat-until* und der *succ*-Funktion auf dem Bildschirm aus.

9. Geben Sie unter Verwendung der *repeat-until*-Anweisung den Umfang eines Kreises in Abhängigkeit vom Kreisradius im Bereich von 23,5 cm bis 43,5 cm auf dem Bildschirm aus! Der Kreisradius wird bei jedem Durchlauf um 0,75 cm erhöht.

5 Vektoren und Matrizen

5.1 Eindimensionale Felder (Vektoren)

Vektoren sind eine strukturierte Zusammenfassung von Variablen *desselben* Datentyps. Auf jedes Vektorelement kann *direkt* mit Hilfe des Index zugegriffen werden.
Die mathematische Darstellung eines Vektors lautet:

$$\vec{a} = \left\{ \begin{array}{c} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{array} \right\}$$

Für die Deklaration eines Vektors wird im Vereinbarungsteil der Bezeichner `array` verwendet. Das nachfolgende Programm verdeutlicht die Deklaration solcher eindimensionalen Vektoren und den direkten Zugriff auf einzelne Vektorelemente:

```
program felder_1;

const obere_grenze=777;

var  zz   : array[1..100] of real;
     gg   : array[1..obere_grenze] of integer;
     name : array[1..134] of char;
     i    : integer;

begin
  for i:=1 to 100 do zz[i]:=random;
    {  i ist der  (Zeilen)- Index }

  { Ausgabe eines bestimmten Elementwertes }
  writeln("Element Nr. 88 :',zz[88]);

  { Ausgabe aller Elementwerte mit for-to-do }
  for i:=1 to 100 do writeln(i,'. Element: ',zz[i]);

readln;
end.
```

Bei der Deklaration eines Vektor müssen die die (Index-) Grenzen direkt vorgegeben werden: Entweder als konkrete Zahl oder im `const`- Abschnitt des Deklarationabschnitts als eine sogenannte Konstante. Die Verwendung von Variablen für die Grenzen eines Arrays ist nicht erlaubt!

5.2 Zweidimensionale Felder (Matrizen)

Matrizen sind - wie die eindimensionalen Felder - eine strukturierte Zusammenfassung von Variablen *desselben* Datentyps. Eine Matrix ist in *Zeilen* und *Spalten* aufgeteilt. Auf jedes Element der Matrix kann *direkt* unter Angabe des Zeilen- und Spaltenindex zugegriffen werden. Die mathematische Darstellung einer Matrix lautet:

$$|a| = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \end{vmatrix}$$

und in allgemeiner Form

$$|a| = \begin{vmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{vmatrix}$$

Die Deklaration erfolgt äquivalent den eindimensionalen Felder unter zusätzlicher Angabe einer zweiten Felddimension:

```
var  feld_zz      : array[1..77,1..88] of real;
     feld_ganz    : array[1..1237,1..3] of integer;
     i,j          : integer;
begin
  {  Zugriff auf das 5. Element in der zweiten Zeile }
i:=2; j:=5;
feld_ganz[i,j]:=45; { 2 =>Zeilenindex 5 =>Spaltenindex }
writeln('Element in Zeile ',i,' und Spalte ',j,' :',feld_ganz[i,j]);

  { Belegen/Initialisieren  aller Feldelemente mit Null}
for i:=1 to 77 do
begin
  for j:=1 to 88 do
  begin
    feld_zz[i,j]:=0;
  end;
end;
  { Belegen/Initialisieren  aller Feldelemente mit Zufallszahlen}
for i:=1 to 77 do
begin
  for j:=1 to 88 do
  begin
    feld_zz[i,j]:=random;
  end;
```

```
end;
end.  { Programmende }
```

Bei der Angabe der Indizes wird zuerst immer der Zeilenindex angegeben! Als Indizes können auch Variablen vom Typ `integer` verwendet, welche auf den gewünschten Wert gesetzt werden.
Die Initialisierung bzw. Belegung aller Elemente einer Matrix erfolgt fast immer unter Verwendung einer *geschachtelten* `for-to-do-` Anweisung. Die verschachtelte `for-to-do-`Anweisung besteht aus einer *äußeren* und einer *inneren* Wiederholungsanweisung (Schleife). In der *äußeren* `for-to-do-` Anweisung wird der Zeilenindex und in der *inneren* Schleife wird der Spaltenindex gesetzt, d. h. die Belegung aller Elemente einer Matrix wird Zeile für Zeile vorgenommen.

5.3 Übungsaufgaben

1. Belegen Sie einen Vektor aus 34 Elementen unter Verwendung der *for-to-do-* Anweisung mit Zufallszahlen!

2. Deklarieren Sie einen Vektor mit 3000 Elementen vom Typ real und belegen Sie jedes Element mit einer Zufallszahl (*random*-Funktion) ! Berechnen Sie die Summe sowie das Maximum und das Minimum über alle Feldelemente unter Verwendung der

   a) *for-to-do*-Anweisung
   b) *repeat-until*-Anweisung
   c) *while-do*-Anweisung

3. Belegen Sie eine Matrix bestehend aus 45 Zeilen und 3 Spalten mit Zufallszahlen und berechnen Sie die Summe der Elemente der 34. Zeile der Matrix unter Verwendung der

   a) *for-to-do*-Anweisung
   b) *repeat-until*-Anweisung
   c) *while-do*-Anweisung.

   Zusatzaufgabe:

   a) Bestimmen Sie das Maximum in jeder Zeile der Matrix !
   b) Bestimmen Sie das Maximum in der Hauptdiagonale der Matrix !

4. Belegen Sie eine Matrix bestehend aus 3 Zeilen und 19 Spalten mit Zufallszahlen und berechnen Sie den Mittelwert der Elemente der 17. Spalte unter Verwendung der

   a) *for-to-do*-Anweisung

b) *repeat-until*-Anweisung

c) *while-do*-Anweisung.

5. Erstellen Sie ein Programm zur Berechnung der Determinante einer Matrix mit

   a) 3 Zeilen und 3 Spalten

   b) 5 Zeilen und 5 Spalten!

6 Deklaration neuer Variablentypen

Mit Hilfe des *type*- Kommandos können im Deklarationsteil neue Variablentypen deklariert werden.

```
type uhu  : array[1..200] of real;
     eule : integer;
var  aa,zz   : uhu;
        i    : eule;
begin
  for i:=1 to 200 do aa[i]:=random;
    {  i ist der  (Zeilen)Index }

  { Ausgabe aller Elementwerte mit for to do }
  for i:=1 to 200 do writeln(i,'. Element: ',aa[i]);
readln;
end.
```

7 Dateiarbeit

7.1 Einleitung

Bisher befanden sich die Daten im Arbeitsspeicher (RAM) und wurden in Form von Variablen abgelegt.Um nun größere Datenbestände zu bearbeiten oder Daten auszutauschen, werden diese Variablen oder Datenbestände auf einem Datenträger (Festplatte, Diskette, ZipDrive) dauerhaft gespeichert.

Im MS-DOS-basierenden Betriebsystemen wird eine Datei durch ihren Namen und das dazughörige Verzeichnis eindeutig gekennzeichnet. Der Zugriff auf eine Datei erfolgt durch die Angabe des Dateinamens.

In Turbo Pascal wird dieser Dateiname durch eine Variable vom Typ *string* angegeben. Der Zugriff auf eine Datei in Form von Lese- und Schreibaktionen wird aber in Turbo Pascal nicht über den Dateinamen vorgenommen, sondern über eine Variable, die bei Textdateien vom Typ *text* und bei typisierten Dateien zum Beispiel vom Typ *real* ist.

Die Zuordnung dieser logischen Dateivariable zu einer physischen Datei muss in Turbo Pascal explizit vorgenommen werden. Dazu wird der Befehl *assign(logische Dateivariable, physischer Dateiname);* verwendet. Das Schreiben und Lesen in bzw. aus einer Datei wird mit den Befehlen *rewrite* und *reset* realisiert. Nach Abschluß dieser Schreib- oder Leseoperationen ist die Datei mit dem Kommando *close* zu schließen! Dieser Vorgang wird im nächsten Abschnitt anhand eines Beispiels erläutert.

7.2 Textdateien

Als Textdateien werden spezielle, nur sequentiell verarbeitbare Dateien bezeichnet, deren Elemente vom Typ CHAR sind.

Der Inhalt dieser Dateien vom Typ *text* kann mit einem gewöhnlichen Editor wie z. B. *notepad* (Windows95) oder dem *edit* (MS-DOS) eingesehen werden.

Zur Markierung von Zeilenende und Dateiende sind spezielle Strukturierungszeichen vorhanden. Diese werden als *end of line* (eoln) und *end of file* (eof) bezeichnet. Mit den oben erwähnten Editoren sind diese Strukturierungszeichen in Form eines Zeilenvorschubes zu erkennen.

Im Gegensatz zum typisierten File (file of real) können die Datensatzlän-gen, d. h. die Längen der einzelnen Elemente variieren.

```
{Arbeiten mit Textfiles}
program BSP3;
const max=5
var uhu : array[1..max] of real;
    i : integer;
    extfile: text;
    extfilename: string;
begin
```

```
  clrscr;
  write('Filename: '); readln(extfilename);
  (* Zuordnung eines Dateinamens zu einer Textdatei *)
  ASSIGN(extfile,extfilename);

  (* Schreiben in eine Datei *)
  REWRITE(extfile);
  for j := 1 to max do
  begin
    write(' uhu(',i,')=');  readln(uhu[i]);
    writeln(uhu[i]);
    writeln(extfile,uhu[i]);
  end;
  CLOSE(extfile);

  (* Lesen aus einer Datei *)
  RESET(extfile);
  for j := 1 to max do
  begin
    readln(extfile,uhu[i]);
    writeln(uhu[i]);
  end;
  CLOSE(extfile);

end.
```

7.3 Aufgaben zur Arbeit mit Textdateien

1. Erzeugen Sie 111 Zufallszahlen vom Typ *real* und schreiben Sie diese Zahlen in eine Datei ! Sehen Sie sich anschließend den Inhalt dieser Datei mit einem Editor an!

2. Sehen Sie sich den Inhalt dieser Datei mit einem Editor im Normal-modus (edit) und in Hex-Modus (edit /80) an! Achten Sie auf die Zeilenende-Zeichen!

3. Lesen Sie den Inhalt dieser Datei wieder ein und berechnen Sie den Mittelwert aller Zahlen!

4. Deklarieren Sie einen Vektor bestehend aus Elementen vom Typ *string*. Belegen Sie diese Vektorelemente mit Inhalten, z. B. Personennamen, und sichern Sie den Vektorinhalt in einer Datei!

5. Lesen Sie den Inhalt der in der vorherigen Aufgabe erzeugten Datei wieder ein und geben Sie ihn auf dem Bildschirm aus!

6. Lesen Sie den Inhalt der in Aufgabe 1 erzeugten Datei als Elemente vom Typ *string* ein und geben Sie ihn auf dem Bildschirm aus!

7. Vergleichen Sie die die Größe der in Aufgabe 1 erzeugten Datei mit der gleichen Anzahl Zahlen vom Typ *real* im Hauptspeicher!

8. Welche Fehler können bei der Arbeit mit Textdateien auftreten?

7.4 Typisierte Dateien

Eine typisierte Datei wird in der Vereinbarungsteil mit *extfile: file of real;* deklariert. File-Variablen dürfen nicht in Ausdrücken oder Ergibtanweisungen verwendet werden, sondern werden ausschließlich als Parameter von Funktionen und Prozeduren (z. B. *writeln(extfile,x);*) verwendet. Bei einer typisierten Datei hat jede Komponente die gleiche Größe (wie bei einem Vektor). Diese Eigenschaft ermöglicht mit den Kommandos *seek* und *filepos* einen direkten und gezielten Zugriff auf die einzelnen Elemente (entsprechend dem Index bei Vektoren). Im nachfolgenden Beispiel erfolgt eine einfache Ein- und Ausgabe in eine typisierte Datei.

```
{Arbeiten mit typisierten Dateien}
program typdatei;
const      max = 5;
var        uhu : array[1..max] of real;
             i : integer;
       extfile : file of real;
    extfilename: string;
begin
  clrscr;
  write('Filename: '); readln(extfilename);
  (* Zuordnung eines Dateinamens zu einer Textdatei *)
  ASSIGN(extfile,extfilename);

  (* Schreiben in eine Datei *)
  REWRITE(extfile);
  for j := 1 to max do
  begin
    write(' uhu(',i,')=');  readln(uhu[i]);
    writeln(uhu[i]);
    write(extfile,uhu[i]);
  end;
  CLOSE(extfile);

  (* Lesen aus einer Datei *)
  RESET(extfile);
  for j := 1 to max do
```

```
  begin
    read(extfile,uhu[i]);
    writeln(uhu[i]);
  end;
  CLOSE(extfile);

end.
```

7.5 Aufgaben zur Arbeit mit typisierten Dateien

1. Erzeugen Sie 111 Zufallszahlen vom Typ *real* und schreiben Sie diese Zahlen in eine typisierte Datei ! Sehen Sie sich anschließend den Inhalt dieser Datei mit einem Editor an!

2. Lesen Sie den Inhalt dieser Datei wieder ein und berechnen Sie den Mittelwert aller Zahlen!

3. Vergleichen Sie die die Größe einer typisierten Datei mit der Größe eines Textfile unter der Voraussetzung einer gleichen Anzahl von Zahlen vom Typ *real*!

4. Welche Fehler können bei der Arbeit mit typisierten Dateien auftreten?

8 Zeichen

Zeichen und Zeichenketten können in Turbo Pascal als Konstanten, als Variablen und als Werte von Ausdrücken auftreten

8.1 Deklaration von Variablen des Typs char

Die Deklaration eines Zeichens kann in Form

- einer Konstante: *const zei1 = 'A';*
- einer Variable: *var zei2: char;*

vorgenommen werden. Das Zeichen muß zwischen zwei Apostrophe ' und ' gesetzt werden.

Bei der Deklaration eines Zeichens wird für diese Variable ein Speicherplatz von 1 Byte im Hauptspeicher (RAM) reserviert. Als Zeichen werden die im ASCII- Zeichensatz (ASCII= American Standard Code for Information Interchange) definierten Zeichen angesehen. Der ASCII-Zeichsatz ist eine Abbildung von alphanumerischen Zeichen auf feste Zahlenwerte. Die Skala reicht von 0 .. 255, wobei die Zahlen bis 127 als 7-Bit ASCII-Zeichensatz und die Zahlen ab 128 als erweiterter 8-bit-ASCII- Zeichensatz bezeichnet werden.

8.2 Der 7-Bit ASCII- Zeichensatz

Dez.	Hex.	Z	Dez.	Hex.	Z	Dez.	Hex.	Z	Dez.	Hex.	Z
0	00	^@	32	20		64	40	@	96	60	‘
1	01	^A	33	21	!	65	41	A	97	61	a
2	02	^B	34	22	"	66	42	B	98	62	b
3	03	^C	35	23	#	67	43	C	99	63	c
4	04	^D	36	24	$	68	44	D	100	64	d
5	05	^E	37	25	%	69	45	E	101	65	e
6	06	^F	38	26	&	70	46	F	102	66	f
7	07	^G	39	27	’	71	47	G	103	67	g
8	08	^H	40	28	(	72	48	H	104	68	h
9	09	^I	41	29	)	73	49	I	105	69	i
10	0A	^J	42	2A	*	74	4A	J	106	6A	j
11	0B	^K	43	2B	+	75	4B	K	107	6B	k
12	0C	^L	44	2C	,	76	4C	L	108	6C	l
13	0D	^M	45	2D	-	77	4D	M	109	6D	m
14	0E	^N	46	2E	.	78	4E	N	110	6E	n
15	0F	^O	47	2F	/	79	4F	O	111	6F	o
16	10	^P	48	30	0	80	50	P	112	70	p

	17	11	^Q		49	31	1		81	51	Q		113	71	q	
	18	12	^R		50	32	2		82	52	R		114	72	r	
	19	13	^S		51	33	3		83	53	S		115	73	s	
	20	14	^T		52	34	4		84	54	T		116	74	t	
	21	15	^U		53	35	5		85	55	U		117	75	u	
	22	16	^V		54	36	6		86	56	V		118	76	v	
	23	17	^W		55	37	7		87	57	W		119	77	w	
	24	18	^X		56	38	8		88	58	X		120	78	x	
	25	19	^Y		57	39	9		89	59	Y		121	79	y	
	26	1A	^Z		58	3A	:		90	5A	Z		122	7A	z	
	27	1B	^[		59	3B	;		91	5B	[		123	7B	{	
	28	1C	^\		60	3C	<		92	5C	\		124	7C	|	
	29	1D	^]		61	3D	=		93	5D	]		125	7D	}	
	30	1E	^^		62	3E	>		94	5E	^		126	7E	~	
	31	1F	^_		63	3F	?		95	5F	_		127	7F		

8.3 Der erweiterte 8-Bit ASCII- Zeichensatz

Dezimal	ASCII	Erklaerung
0	NUL	null
1	SOH	start of heading
2	STX	start of text
3	ETX	end of text
4	EOT	end of transmission
5	ENQ	enquiry
6	ACK	acknowledge
7	BEL	bell
8	BS	backspace
9	HT	horizontal tab
10	LF	new line
11	VT	vertical tab
12	FF	form feed
13	CR	carriage return
14	SO	shift out
15	SI	shift in
16	DLE	data link escape
17	DC1	device control 1
18	DC2	device control 2
19	DC3	device control 3
20	DC4	device control 4
21	NAK	negative acknowledge
22	SYN	synchronous idle

23	ETB	end of transmission block	
24	CAN	cancel	
25	EM	end of medium	
26	SUB	substitute	
27	ESC	escape	
28	FS	file separator	
29	GS	group separator	
30	RS	record separator	
31	US	unit separator	
32	SP	digit select	
33	!	exclamation point	
34	"	double quotation mark	
35	#	pound sign, number sign	
36	$	dollar sign	
37	%	percent sign	
38	&	ampersand	
39	’	apostrophe	
40	(	left parenthesis	
41	)	right parenthesis	
42	*	asterisk	
43	+	addition sign	
44	,	comma	
45	-	subtraction sign	
46	.	period	
47	/	right slash	
48	0		
49	1		
50	2		
51	3		
52	4		
53	5		
54	6		
55	7		
56	8		
57	9		
58	:	colon	
59	;	semicolon	
60	<	less than	
61	=	equal	
62	>	greater than	
63	?	question mark	
64	@	at sign	
65	A		
66	B		

67	C		
68	D		
69	E		
70	F		
71	G		
72	H		
73	I		
74	J		
75	K		
76	L		
77	M		
78	N		
79	O		
80	P		
81	Q		
82	R		
83	S		
84	T		
85	U		
86	V		
87	W		
88	X		
89	Y		
90	Z		
91	[	left bracket	
92	\	left slash, backslash	
93	]	right bracket	
94	^	hat, circumflex, caret	
95	_	underscore	
96	`	grave accent	
97	a		
98	b		
99	c		
100	d		
101	e		
102	f		
103	g		
104	h		
105	i		
106	j		
107	k		
108	l		
109	m		
110	n		

```
|111          |o          |                                    |
|112          |p          |                                    |
|113          |q          |                                    |
|114          |r          |                                    |
|115          |s          |                                    |
|116          |t          |                                    |
|117          |u          |                                    |
|118          |v          |                                    |
|119          |w          |                                    |
|120          |x          |                                    |
|121          |y          |                                    |
|122          |z          |                                    |
|123          |{          |left brace                          |
|124          ||          |logical or, vertical bar            |
|125          |}          |right brace                         |
|126          |~          |similar, tilde                      |
|127          |DEL        |delete                              |
=============================================================
```

8.4 Arbeiten mit Variablen vom Typ char

Entsprechend der Reihenfolge der im ASCII- Zeichensatz aufgeführten Zeichen können mit den Kommandos *pred* und *succ* jeweils der Vorgänger und der Nachfolger ermittelt werden. Folgendes Programm schreibt die Kleinbuchstaben von *a* bis *x*, getrennt durch ein Leerzeichen, auf den Bildschirm.

```
program kleinbuchstaben;
uses crt;
var  vor,nach:char;

{=== Beginn des Hauptprogrammes ===}
begin
clrscr;   { Bildschirm loeschen }

nach:='a';
repeat
  vor:=nach;
  write(vor,'  ');
  nach:=succ(vor);
until  vor='x'

readln;
end.
```

Ebenso können die Zeichen mit dem Dezimalcode ausgegeben werden. Dazu wird die *chr*-Funktion verwendet. Die *chr*-Funktion wandelt eine vorgebene Dezimalzahl entsprechend dem ASCII-Zeichensatz in das dazugehörige ASCII-Zeichen um. Das folgende Programm schreibt die Großbuchstaben von *B* bis *T*, getrennt durch ein Leerzeichen, auf den Bildschirm.

```
program grossbuchstaben;
uses crt;
var i:integer;

{=== Beginn des Hauptprogrammes ===}
begin

clrscr;   { Bildschirm loeschen }

for i:= 66 to 84 do
begin
  write('Dezimal: ',i,' ASCII: ',chr(i),' ');
end;

readln;
end.
```

Die *ord*-Funktion dagegen wandelt ein vorgebenes ASCII- Zeichen entsprechend dem ASCII-Zeichensatz in den dazugehörigen Dezimalcode um. Das folgende Programm schreibt den Dezimalcode der Buchstaben von *B* bis *T*, getrennt durch ein Leerzeichen, auf den Bildschirm:

```
program dezimal;
uses crt;
var a:char;

{=== Beginn des Hauptprogrammes ===}
begin

clrscr;   { Bildschirm loeschen }

for a:= 'B' to 'T' do
begin
  writeln('Zeichen: ',a,' Dezimal: ',ord(a),' '):
end;

readln;
end.
```

8.5 Übungsaufgaben

1. Schreiben Sie ein Programm, welches die Zeichen von k bis y entsprechend dem ASCII-Zeichensatz ausgibt! Arbeiten Sie mit dem Dezimalcode dieser Zeichen!

2. Schreiben Sie ein Programm, welches den Dezimalcode der Zeichen von S bis X ausgibt!

3. Schreiben Sie ein Programm, welches das gesamte Alphabet in Großbuchstaben ausgibt!

4. Geben Sie die Zahlen von 1 bis 10 als Variable vom Typ *char* bzw. als ASCII-Zeichen aus! Verwenden Sie dazu die *chr*-Funktion!

9 Zeichenketten (String)

9.1 Deklaration von Strings

Zeichenketten (string) stellen in Turbo Pascal eine feste unlösbare Aneinanderreihung von Zeichen (char) dar. Nachstehende Abbildung veranschaulicht die Struktur eines Strings besthend aus fünf Buchstaben (Zeichen):

Zeichen	L	i	n	d	e
Index	1	2	3	4	5

Die Deklaration von Zeichenketten erfolgt mit dem Bezeichner *string* oder *string[20]*:

```
var s1,s2,x:string;
```

Ein String kann somit als ein Vektor (`array[1..255] of char`) - dessen Elemente mit Variablen vom Typ `char` gefüllt sind - angesehen werden. Auf jedes Zeichen in einem String kann unter Verwendung des Index direkt zugegriffen werden:

```
  var s1:string;
       i: integer;
begin
  s1:='Die Sonne scheint';
  writeln('Der 5. Buchstabe ist ein: ',s1[5]);
  writeln('Der ganze String: ',s1);

  { Jetzt mit for to do jedes Zeichen einzelnd ausgeben }
  for i:=1 to length(s1) do write(s1[i]);
  { Was macht die Funktion length ? )
end.
```

9.2 Ein einfaches Programmbeispiel

Die Ein- und Ausgabe von Zeichenketten ist im nachfolgenden Programm in einfacher Form dargestellt:

```
program zeichenketten;
  uses crt;
  var s1:string;  { max. 255 Zeichen ! }
begin
  clrscr;   { Bildschirm loeschen }
  Write('Geben Sie bitte Ihren Vornamen ein: ');  readln(s1);
  writeln('Ihr Vorname ist: ',s1);
  readln;
end.
```

Die Deklaration der Variable *s1* als Variablentyp *string* bedeutet, das der String eine maximale Länge von 255 Zeichen besitzen darf. Die Deklaration der Variable *s1* als Variablentyp *string[10]* dagegen begrenzt die maximale Länge der eingegebenen Zeichenkette auf maximal 10 Zeichen.

9.3 Zeichenkettenoperationen

9.3.1 Funktionen

Länge einer Zeichenkette Wie kann man die Länge eines mit Inhalten belegten Strings feststellen? Dazu verwendet man die Funktion **length**:

```
program zeichenketten;
uses crt;
var s1:string;

{=== Beginn des Hauptprogrammes ===}
begin

clrscr;   { Bildschirm loeschen }
write('Geben Sie bitte Ihren Vornamen ein: ');

readln(s1);
writeln('Ihr Vorname ist: ',s1);
writeln('Ihr Vorname besteht aus ',length(s1),' Zeichen');
readln;
end.
```

Anmerkung: Mit der Funktion **length** wird nicht die Länge des Strings entsprechend der Deklaration im Variablenteil bestimmt, sondern die Anzahl der mit Zeichen (char) belegten Stringelemente!

Verbinden von Zeichenketten Mehrere Strings könne miteinander zu einen String verbunden werden. Das nachfolgende Programm zeigt das Verbinden von der Zeichenketten **s1** und **s2** zu einem neuen String **s3** mit der **concat**- Funktion.

```
program zeichenketten;
uses crt;
var s1,s2,s3:string;
           l: integer;

{=== Beginn des Hauptprogrammes ===}
begin

clrscr;   { Bildschirm loeschen }
```

```
Write('Geben Sie bitte Ihren Vornamen ein: ');   readln(s1);
Write('Geben Sie bitte Ihren Nachnamen ein: ');   readln(s2);
s3:=s1+s2;
writeln('Ihr ganzer Name ist: ',s3);
l:=length(s3);
writeln('Ihr Name besteht aus ',l,' Zeichen');

{ Eine zweite Moeglichkeit }
s3:=concat(s1,s2);
writeln('Ihr ganzer Name ist: ',s3);
l:=length(s3);
writeln('Ihr Name besteht aus ',l,' Zeichen');

write('Ihr Name rueckwaerts: ');
for i:=l downto 1 do write s3[i];
writeln;
writeln('Bitte druecken Sie die Enter-Taste !');
readln;
end.
```

Anmerkung: Der + Operator kann gleichwertig zu der **concat**- Funktion verwendet werden.

Suchen in Zeichenketten Mit Hilfe der Funktion **pos** ist es möglich, in einer gegebenen Zeichenkette **s1** die Position **position** eines bestimmten Suchmusters **muster** zu ermitteln.

```
program zeichenketten;
uses crt;
var    s1,muster : string;
        position : integer;

{=== Beginn des Hauptprogrammes ===}
begin

clrscr;   { Bildschirm loeschen }
Write('Geben Sie bitte Ihren Vornamen ein: ');  readln(s1);
Write('Geben Sie bitte das Suchmuster ein: ');  readln(muster);
position:=pos(muster,s1);
writeln('Erstes Auftreten des Suchmusters: ',position);

writeln;
writeln('Bitte druecken Sie die Enter-Taste !');
readln;
end.
```

Kopieren aus Zeichenketten Mit Hilfe der Funktion **copy** ist es möglich, aus einer gegebenen Zeichenkette **s1** ab der Position **anfang** eine Zeichenkette mit der Länge **laenge** in einen zweiten String **s2** zu kopieren.

```
program zeichenketten;
uses crt;
var        s1,s2 : string;
   anfang,laenge : integer;

{=== Beginn des Hauptprogrammes ===}
begin

clrscr;   { Bildschirm loeschen }
Write('Geben Sie bitte Ihren Vornamen ein: ');    readln(s1);
Write('Laenge des zu kopierenden Stringteils: ');
  readln(laenge);
Write('Beginn des zu kopierenden Stringteils: ');
  readln(anfang);

s2:=copy(s1,anfang,laenge);
writeln('Inhalt des Strings s2: ',s2);

writeln;
writeln('Bitte druecken Sie die Enter-Taste !');
readln;
end.
```

9.3.2 Prozeduren

Einfügen von Zeichenketten Mit der Prozedur **insert(s1,s2,position)** wird in den String **s2** nach dem Zeichen mit dem Index **position** der String **s1** eingefügt.

```
program zeichenketten;
uses crt;
var s1,s2,s3:string;
           k: integer;

{=== Beginn des Hauptprogrammes ===}
begin

clrscr;   { Bildschirm loeschen }
s1:='Es wird regnen.');
s2:='nicht ';
insert(s2,s1,9);
  { Ausgabe des Stringinhaltes }
```

```
writeln(s1);
k:=length(s1);
writeln('Der String s1 besteht aus ',k,' Zeichen');

write('Stringausgabe rueckwaerts: ');
for i:=k downto 1 do write(s1[i]);
writeln;
writeln('Bitte druecken Sie die Enter-Taste !');
readln;
end.
```

Löschen in Zeichenketten Mit der Prozedur *delete(s1,position,anzahl)* werden in den String **s1** ab dem Zeichen mit dem Index **position** genau so viele Zeichen gelöscht, wie die Variable **anzahl** angibt.

```
program zeichenketten;
uses crt;
var s1,s2,s3:string;
    k,anzahl,position: integer;

{=== Beginn des Hauptprogrammes ===}
begin

clrscr;   { Bildschirm loeschen }
s1:='Heute ist kein heisser Tag!';
k:=length(s1);
writeln('s1 besteht vor dem Loeschen aus ',k,' Zeichen');

postion:=15; anzahl:=8;
delete(s1,position,anzahl);
  { Ausgabe des Stringinhaltes }
writeln(s1);
k:=length(s1);
writeln('s1 besteht nach dem Loeschen aus ',k,' Zeichen');

write('Stringausgabe rueckwaerts: ');
for i:=k downto 1 do write(s1[i]);
writeln;
writeln('Bitte druecken Sie die Enter-Taste !');
readln;
end.
```

Umwandlung von Zahlen in Zeichenketten Mit der Prozedur **str** ist möglich, eine Variable vom Typ *integer* oder *real* in eine Variable vom Typ *string* umzuwandeln.

```
program umwandlung_zahl_in_string;
uses crt;
var s1  : string;
    k,i : integer;
    r   : real;

{=== Beginn des Hauptprogrammes ===}
begin

clrscr;   { Bildschirm loeschen }

{========Beispiel 1=======}
k:=1533;
str(k,s1);
writeln(s1);

{========Beispiel 2=======}
r:=1.5426e-12;
str(r,s1);
writeln(s1);

{========Beispiel 3=======}
r:=-0.004382;
str(r,s1);
writeln(s1);

writeln;
writeln('Bitte druecken Sie die Enter-Taste !');
readln;
end.
```

Umwandlung von Zeichenketten in Zahlen Mit der Prozedur **val** ist möglich, die Variable **s1** vom Typ *string* in die Variable **k** vom Typ *integer* oder *real* umzuwandeln.

```
program umwandlung_string_in_zahl;
uses crt;
var s1       : string;
    k,fehler : integer;
    r        : real;

{=== Beginn des Hauptprogrammes ===}
begin

clrscr;   { Bildschirm loeschen }
```

```
writeln('Geben Sie bitte Ihre Telefonnummer ein: ');
readln(s1);

val(s1,k,fehler);
if fehler = 0 then
begin
  writeln('Ihre Telefonnummer ist: ',k);
end
else
begin
  writeln('Fehler in der Eingabe');
end;

writeln;
writeln('Bitte druecken Sie die Enter-Taste !');
readln;
end.
```

Anmerkung: Bei der Arbeit mit der val - Funktion sollte immer der Fehlercode abgefragt werden. Nur wenn der Fehlercode null ist, dann kann die Konvertierung durchgeführt werden! Bei einem Fehlercode ungleich null enthält der eingegebene String dann ASCII-Zeichen, welche nicht den Zeichen von 1..9 entsprechen.

9.4 Übungsaufgaben

1. Schreiben Sie ein Programm, welches die Länge des eingebenen Strings berechnet und auf dem Bildschirm ausgibt!

2. Geben Sie den Inhalt des in Aufgabe 1 eingegebenen Strings in umgekehrter Reihenfolge auf dem Bildschirm aus!

3. Schreiben Sie ein Programm, welches den Inhalt von vier Variablen des Typs *string* einliest und miteinander verbindet! Arbeiten Sie mit der *concat*-Funktion! Aus wieviel Zeichen besteht der neue String?

4. Fügen Sie in den in Aufgabe 1 erzeugten String nach jedem Buchstaben mit Hilfe der *insert*- Funktion ein Leerzeichen ein! Bestimmen Sie die Länge der modifizierten Zeichenkette!

5. Lesen Sie eine Zeichenkette ein und löschen Sie in dieser Zeichenkette das erste und letzte Zeichen!

6. Lesen Sie eine Zeichenkette mit einer Länge von mindestens 10 Zeichen ein und vertauschen Sie das erste und letzte Zeichen!

7. Lesen Sie eine Zeichenkette ein und untersuchen Sie diese Zeichenkette (bzw. jedes Element dieser Zeichenkette) auf die Existenz des Zeichens *e* oder *i* oder anderer Zeichen!

8. Wandeln Sie einen String in eine Zahl vom Typ *integer* um! Berechnen Sie von der Zahl die Quadratwurzel!

9. Wandeln Sie einen String in eine Zahl vom Typ *real* um! Geben Sie den String wie eine real- Variable ein - mit Mantisse und Exponent.

10. Konvertieren Sie eine eingebene Zahl vom Typ *real* in einen String! Achten Sie dabei auf die Mantisse und den Exponenten.

10 Unterprogramme

10.1 Was sind Prozeduren und Funktionen?

Prozeduren und Funktionen (Blöcke) dienen der logischen Unterteilung des Programmes (Quelltextes). Solch ein Unterprogramm (procedure) besteht wie ein Hauptprogramm aus einem Vereinbarungsteil und einem Anweisungsteil.

Eine Prozedur besteht in Analogie zum allgemeinen Programmaufbau aus *Vereinbarungsteil* und *Anweisungsteil*. Der prinzipielle Aufbau eines Pascalprogrammes, welches eine Prozedur namens *musik* enthält, ist im nachfolgenden Quelltext dargestellt:

```
program Arbeiten_mit_Prozeduren; { Arbeiten mit Prozeduren }
uses crt;                        { Einbinden der Unit crt }
var i:integer;

procedure musik; {Anfang der Prozedur}
begin
  for i:=1 to 10 do writeln('Der ',i,'. Ton',#7);
  writeln('Bitte druecken Sie die Enter-Taste !');
  readln;
end;
{ Ende der Prozedur }

{ Begin des Hauptprogrammes }
begin
  clrscr;  { Loeschen des Bildschirmes }
  musik;   { <<<<< Aufruf der Prozedur mit dem Namen Musik }
end.
```

Die Funktion (function) stellt in Pascal einen Spezialfall einer Prozedur dar und enthält nur *einen* Outputparameter (siehe nächsten Abschnitt).

10.2 Die Parameter einer Prozedur

Jede Prozedur und Funktion benötigt Input- und Outputparameter. Wenn bei einem Rechteck aus den gegebenen Seitenlängen a und b der Umfang U und die Fläche A des Rechtecks berechnet werden sollen, dann stellen a und b die Inputparameter und U und A die Outputparameter bzw. -variablen dar.

Die in der Prozedur benötigten Variablen können als *globale* Variablen im Deklarationsteil des Hauptprogramms vereinbart werden. Das Arbeiten mit *globalen* Variablen kann schnell zu Fehlern im Programmablauf führen, weil die im Hauptprogramm verwendeten Variablen direkt in den Prozeduren geändert werden können. Deshalb sollte beim Programmieren immer mit *lokalen* Variablen gearbeitet werden (Abschnitt 1.2.2).

10.2.1 Prozedur mit globalen Variablen

Das nachfolgende Programmbeispiel enthält eine Prozedur, deren Aufruf die Übergabe eines Parameter notwendig macht. Der Aufruf erfolgt dann in diesem Beispiel mit *musik;*.

```
{ Arbeiten mit Prozeduren und Funktionen }
program Arbeiten_mit_Prozeduren;
uses crt;
var i,anzahl: integer;
{ i und anzahl sind globale Variablen }

{ Hier erfolgt die Deklaration der Prozedur}
procedure musik;  { <<<<< Start des Unterprogramms }
begin
  for i:=1 to anzahl do writeln('Der ',i,'. Ton',#7);
  writeln('Bitte druecken Sie die Enter-Taste !');
  readln;
end;
{ Ende der Prozedur }

{ Begin des Hauptprogrammes }
begin
  clrscr;  { Loeschen des Bildschirmes }

  writeln('Jetzt hoeren Sie 5 Toene !');
  anzahl:=5;
  musik;           { <<<<< Musik mit 5 Toenen }

  write('Wieviel Toene moechten Sie hoeren ? ..: ');
  readln(anzahl);
  musik;           { <<<<< Prozeduraufruf }
end.
```

Die Variablen mit den Namen *anzahl* und *i* sind in diesem Programmbeispiel im Deklarationsteil des Hauptprogrammes vereinbart und werden damit als *globale* Variablen bezeichnet.

10.2.2 Prozedur mit lokalen Variablen

Das nachfolgende Programmbeispiel enthält eine Prozedur, deren Aufruf die Übergabe eines Parameter notwendig macht. Zusätzlich wird hier mit lokalen Variablen gearbeitet. Das Arbeiten mit lokalen Variablen ist aus der Tatsache ersichtlich, daß im *Prozedurkopf* sogenannte Übergabevariablen definiert werden. Der Aufruf erfolgt dann in diesem Beispiel mit *musik(anzahl);*.

```
{ Arbeiten mit Prozeduren und Funktionen }
program Arbeiten_mit_Prozeduren;
uses crt;
var anzahl: integer;
      { nur (!) Variable anzahl hier als globale Variable}

{ Hier erfolgt die Deklaration der Prozedur}
procedure musik(gardine: integer);     { <<<<< Start des Unterprogramms }
  var i:integer; { dieses i ist nur in der Prozedur gueltig! }
begin
  for i:=1 to  gardine do writeln('Der ',gardine,'. Ton',#7);
  writeln('Bitte druecken Sie die Enter-Taste !');
  readln;
end;
{ Ende der Prozedur }


begin       { Begin des Hauptprogrammes }
  clrscr;   { Loeschen des Bildschirmes }

  writeln('Jetzt hoeren Sie 5 Toene !');
  musik(5);   {  <<<<< Musik mit 5 Toenen }

  write('Wieviel Toene moechten Sie hoeren ? ..: ');
  readln(anzahl);
  musik(anzahl);   { <<<<< Prozeduraufruf mit dem Parameter anzahl}
end.
```

10.2.3 Ein Programmbeispiel

Das nachfolgende Programmbeispiel enthält eine Prozedur zur Berechnung einer Kreisfläche. In diesem Pascalprogramm wird streng mit lokalen Variablen gearbeitet. Der Aufruf erfolgt mit *kreisflaeche(radius,flaeche,umfang);*.

```
{ Arbeiten mit Prozeduren und Funktionen }
program Arbeiten_mit_Prozeduren;
uses crt;
   { Diese Variablen nur fuer das Hauptprogramm !!}
var radius,umfang, flaeche: integer;

{ Hier erfolgt die Deklaration der Prozedur}
procedure kreisflaeche(rr:real; var umf,fl:real);
begin              { rr, umf und fl sind lokale Variablen }
  umf:=2*pi*rr;
```

```
  fl:=pi*rr*rr;
end;
{ Ende der Prozedur }

{ Begin des Hauptprogrammes }
begin
  clrscr;  { Loeschen des Bildschirmes }
  writeln('Programm zur Berechnung einer Kreisflaeche');
  writeln('=======================================');
  write('Geben Sie den Radius ein: ');
  readln(radius);

  { jetzt erfolgt der Aufruf der Prozedur kreisflaeche }
  { ================================================ }
  kreisflaeche(radius, umfang,flaeche);
  { ================================================ }

  { jetzt erfolgt die Ausgabe des Ergebnisses }
  writeln(' Der Kreisumfang betraegt: ',umfang);
  writeln(' Die Kreisflaeche betraegt: ',flaeche);

  writeln('Bitte druecken Sie die Enter-Taste !');
  readln;
end.
```

10.3 Funktionen (function)

10.3.1 Was sind Funktionen?

Wie schon erwähnt, stellt die Funktion (*function*) in Turbo Pascal einen Spezialfall einer Prozedur dar. Die Funktion hat nur *einen Ausgabeparameter*. So kann aus den den Seitenlängen a und b eines Rechtecks der Umfang mit Hilfe einer Funktion berechnet werden.

10.3.2 Ein Programmbeispiel

```
program Arbeiten_mit_Funktionen;
uses crt;
   { Diese Variablen nur fuer das Hauptprogramm !!}
var a,b,umfang : integer;

{ Hier erfolgt die Deklaration der Funktion}
function rechteck(axt,beil:real):real;
begin              { axt und beil sind lokale Variablen }
```

```
  rechteck:=2*(axt+beil);
end;
{ Ende der Funktion }

{ Begin des Hauptprogrammes }
begin
  clrscr;  { Loeschen des Bildschirmes }
  writeln('Programm zur Berechnung einer Kreisflaeche');
  writeln('========================================');
  write('Geben Sie die laenge der Seite a ein: ');
  readln(a);
  write('Geben Sie die laenge der Seite b ein: ');
  readln(b);

  { jetzt erfolgt der Aufruf der Funktion names rechteck }
  umfang:=rechteck(a,b);

  { jetzt erfolgt die Ausgabe des Ergebnisses }
  writeln(' Der Umfang betraegt: ',umfang);

  { Der Funktionsaufruf kann auch direkt in eine Anweisung }
  { eingebunden werden !}
  writeln(' Der Umfang betraegt: ',rechteck(a,b));

  writeln('Bitte druecken Sie die Enter-Taste !');
  readln;
end.
```

10.4 Übungsaufgaben

1. Erstellen Sie ein Programm unter Verwendung einer Prozedur zur Ausgabe einer Bildschirmzeile mit 80 Gleichheitszeichen!
2. Erstellen Sie ein Programm, welches ein Unterprogramm

   a) in Form einer Funktion *function*
   b) in Form einer Prozedur *procedure*

   zur Berechnung der Quadratwurzel mit der *sqrt*-Funktion enthält!
3. Erstellen Sie ein Programm unter Verwendung einer Prozedur zur Berechnung des *Minimums* und *Maximums* von *n* Zufallszahlen (Input: Anzahl n; Output: min, max)!
4. Erstellen Sie ein Pascalprogramm, welches ein Unterprogramm zur Berechnung folgender Kugelparameter enthält:

- Input: Kugelradius
- Output: Kugelvolumen $V_K = \frac{4}{3} \cdot \pi \cdot r^3$ und Kugelfläche $A_K = 4 \cdot \pi \cdot r^2$

Arbeiten Sie im Unterprogramm (*procedure*) mit lokalen Variablen und im Hauptprogramm mit globalen Variablen.

5. Erstellen Sie ein Programm unter Verwendung einer Prozedur zur Berechnung der Summe von n Zufallszahlen (Input: Anzahl n; Output Summe s)!

6. Erstellen Sie ein Pascalprogramm, welches ein Unterprogramm zur Lö-sung folgender Gleichung enthält: $x_{1,2} = -\frac{a}{2} \pm \sqrt{\frac{a^2}{4} - b}$

   - Input: a, b
   - Output: x_1, x_2

   Arbeiten Sie im Unterprogramm (*procedure*) mit lokalen Variablen und im Hauptprogramm mit globalen Variablen! In der Prozedur erfolgt nur die Berechnung von x_1 und x_2. Die Ein- und Ausgabe wird im Hauptprogramm vorgenommen!

7. Erstellen Sie ein Pascal Programm unter Verwendung einer Prozedur zur Bestimmung des Schnittpunktes von zwei Geraden!

   Input:

   a) Gerade 1: Punkt 1: x_{11}, y_{12}; Punkt 2: x_{21}, y_{22}
   b) Gerade 2: Punkt 3: x_{31}, y_{32}; Punkt 4: x_{41}, y_{42}

   Output: Schnittpunkt x_{sp}, y_{sp}

11 Verbundvariablen (record)

11.1 Was sind Verbundvariablen?

Mit dem Datentyp *record* lassen sich Variablen unterschiedlichen Typs in einer Struktur zusammenfassen. Ein *record* ist eine Datenstruktur mit einer festen Anzahl von Komponenten. Diese Komponenten werden auch als Felder bezeichnet und stellen eine Variable beliebigen Typus (integer, real, string, boolean, char etc.) dar. Beispiele dafür sind Datenbankeinträge in Form von Adreßeinträgen oder Lagerbeständen und in einfacher Form komplexe Zahlen bestehend aus Real- und Imaginärteil. Mit Hilfe einer Struktur werden zusammengehörige Daten zusammengefaßt.

In nachfolgender Abbildung ist eine Struktur zur Charakterisierung eines Autos dargestellt:

Typ	Farbe	Alter	Preis
string	string	integer	real

In der ersten Zeile der Tabelle sind die Parameter eines Autos in Form von Variablen angegeben und in der zweiten Tabellenzeile sind die dazugehörigen Variablentypen angegeben. Bei der Definition für eine Struktur muß für jede Komponente deren Namen und Typ angegeben werden.

Im nachfolgenden Quelltext ist in einfacher Form der Umgang mit Variablen vom Typ *record* dargestellt.

11.2 Die Deklaration von record- Variablen

Im nachfolgenden Programmausschnitt ist die typische Abfolge einer Deklaration einer *record*- Variablen dargestellt. Im *type*- Vereinbarungsteil wird zunächst eine neuer Variablentyp *auto* definiert, hinter welcher sich eine *record*- Struktur verbirgt. Diese *record*-Struktur besteht aus drei Variablen unterschiedlichen Typs: *string*, *real* und *integer*.

Im *var*- Abschnitt erfolgt dann die endgültige Deklaration der Variablen *opel* und *ford* vom Variablentyp *auto*.

```
type auto = record
              farbe: string;
              preis: real;
              alter: integer;
            end;

var opel,ford: auto;
```

Es ist möglich, eine *record* Struktur im *var*- Abschnitt direkt zu deklarieren. Die Vereinbarung im *type*- Abschnitt führt zu einer besseren Strukturierung des Quelltextes.

11.3 Das Programmieren mit record- Variablen

11.3.1 Ein einfaches Beispiel

Im nachfolgen Pascalprogramm wird mit einer Struktur namens *auto* gearbeitet. Jede Komponente dieser Struktur wird mit Inhalten belegt und anschließend wieder ausgegeben.

```
program Einfaches_record_Beispiel;
type auto = record
              farbe: string;
              preis: real;
              alter: integer;
            end;

var meins, deins: auto;

begin
  writeln('Eine kleine Autodatenbank');
  writeln('Das erste Auto:');
  write('Farbe des Autos: ');readln(meins.farbe);
  write('Preis des Autos: ');readln(meins.preis);
  write('Alter des Autos: ');readln(meins.alter);

  { Das war die erste Moeglichkeit - Jetzt das with Kommando }
  with meins do
  begin
    write('Farbe des Autos: ');readln(farbe);
    write('Preis des Autos: ');readln(preis);
    write('Alter des Autos: ');readln(alter);
  end;

  { Mit der Variable deins koennten wir dasselbe machen }
  writeln('Das zweite Auto:');
  write('Farbe des Autos: ');readln(deins.farbe);
  write('Preis des Autos: ');readln(deins.preis);
  write('Alter des Autos: ');readln(deins.alter);

  write('Farbe des 1. Autos: ',meins.farbe);
  write('Farbe des 2. Autos: ',deins.farbe);

  deins:=meins;
    { Uebergabe eines gesamten Records ist erlaubt !!! }

  writeln('Farbe des Autos: ',deins.farbe);
```

```
  writeln('Farbe des Autos: ',deins.preis);
  writeln('Farbe des Autos: ',deins.alter);
end.
```

11.3.2 record- Variablen am Beispiel einer Adreßdatenbank

```
{Arbeiten mit strukturierten Datentypen - record}
program Struktur1;
uses crt;
type mensch = record
                          vorname:string;
                          nachname:string;
                          alter:integer;
                          strasse:string;
                          hausnummer:integer;
                        end;

var student : array[1..1000] of mensch;
          i : integer;

begin
  clrscr;
  writeln('Programm zur Eingabe von Adressen');

for i:=1 to 5000 do with student[i] do
begin
  write('Eingabe des Vornamens ein: ');readln(vorname);
  write('Eingabe des Nachnamens ein: ');readln(nachname);
  write('Eingabe des Alters ein: ');readln(alter);
  write('Eingabe der Strasse ein: ');readln(strasse);
  write('Eingabe der Hausnummer ein: ');readln(hausnummer);
end;

for i:=1 to 5000 do with student[i] do
begin
    writeln('Vorname: ',vorname);
    writeln('Nachname: ',nachname);
    writeln('Alter: ',alter);
    writeln('Strasse: ',strasse);
    writeln('Hausnummer: ',hausnummer);
end;


end.
```

Diese *array*- Variante ist aber insbesondre im Hinblick auf das Speichermanagement sehr statisch - nichtbenutzte Elemente benötigen wertvollen Speicherplatz z. B. bei nichtbelegten Adreßeinträgen. Eine wesentlich elegantere und gebräuchlichere Variante stellen Listen dar.

11.4 Übungsaufgaben

1. Erzeugen Sie eine kleine Adreßdatenbank unter Verwendung von *record*-Variablen mit mindestens fünf Adreßeinträgen und belegen Sie diese Struktur mit Inhalten unter Verwendung von Hilfsvariablen!

2. Speichern Sie diese fünf Adressen in einer Datei vom Typ *text* und sehen Sie sich den Inhalt dieser Datei mit dem Pascal-Editor an. Wie groß ist diese Datei (in kByte)?

3. Erzeugen Sie eine kleine Warenbestandsliste unter Verwendung von *record*-Variablen mit mindestens 4 verschiedenen Artikeln. Diese Artikel sind jeweils durch Bezeichnung, Preis, Anzahl und Farbe charakterisiert. Speichern Sie die Warenbestandsliste in einer Datei vom Typ *text* ab und geben Sie die Größe dieser Datei an!

4. Versuchen Sie diese Warenbestandsliste in einer typisierten Datei abzulegen! Wie groß ist diese Datei?

5. Vergleichen Sie die Größe der in Aufgabe 4 erzeugten Datei mit der in Aufgabe 3 erzeugten Datei. Wie erklären Sie sich die unterschiedlichen Dateigrößen?

6. Definieren Sie diese in Aufgabe 1 erzeugten *record*-Variablen als dynamische Variablen!

7. Definieren Sie sich eine zweite dynamische Variable mit der in Aufgabe 1 verwendeten Struktur und belegen Sie die einzelnen record-Elemente mit Inhalten!

12 Datenformate

12.1 Ganzzahlige Variablentypen

Folgende Variablentypen stehen zur Arbeit mit ganzen Zahlen zur Verfügung:

1. shortint: 2 Bytes (-128..+127) (1 Bit als Vorzeichen)

2. byte: 1 Bytes (0..255) (kein Vorzeichen, immer positiv)

3. integer: 2 Bytes (-32768..+32767) (1 Bit als Vorzeichen)

4. Word: 2 Bytes (0..65535) (kein Vorzeichen, immer positiv)

5. longint: 4 Bytes (1 Bit als Vorzeichen)

12.2 Gleitkommazahlen

Bei der Deklaration einer Variable vom Typ *real* wird ein Speicherplatz von 6 Bytes (48 Bit) belegt. Der Datentyp *real* wird in Form von Mantisse und Exponent dargestellt: $\pm 1.6578..45E+01$. Diese Form der Darstellung gilt für sämtliche Gleitkomma-Variablentypen, wie *real, single, double, extended*. Der Dezimalpunkt wird nicht explizit gespeichert, sondern implizit angenommen. Es ergeben sich für die unterschiedlichen Gleitkomma-Variablentypen auch unterschiedliche Speicherplatzanforderungen und damit auch unterschiedliche Zahlenbereiche:

1. single: 4 Bytes
2. real: 6 Bytes
3. double: 8 Bytes
4. extended: 10 Bytes

12.3 Aufgaben zu den Datenformaten

1. Bestimmen Sie unter Verwendung des Turbo Pascal Hilfesystems die Zahlenbereiche für jeden Gleitkomma- Variablentyp!
2. Deklarieren Sie einen Vektor mit 3000 Elementen vom Typ *real* und belegen Sie jedes Element mit einer Zufallszahl (*random*-Funktion) ! Wieviel Speicherplatz benötigen alle Elemente des Vektor zusammen (in Bytes !) ?
3. Speichern Sie die in Aufgabe 2 erzeugten Zufallszahlen in einer *typisierten* Datei! Wieviel Speicherplatz belegt diese Datei auf der Festplatte (in Bytes)?
4. Speichern Sie die in Aufgabe 2 erzeugten Zufallszahlen in einer Datei vom Typ *text* ab! Wieviel Speicherplatz belegt diese Datei auf der Festplatte (in Bytes)?
5. Wie erklärt sich der unterschiedliche Speicherplatzbedarf von typisierten und von Textdateien?

13 Datenformate

13.1 Ganzzahlige Variablentypen

Folgende Variablentypen stehen zur Arbeit mit ganzen Zahlen zur Verfügung:

1. shortint: 2 Bytes (-128..+127) (1 Bit als Vorzeichen)
2. byte: 1 Bytes (0..255) (kein Vorzeichen, immer positiv)
3. integer: 2 Bytes (-32768..+32767) (1 Bit als Vorzeichen)
4. Word: 2 Bytes (0..65535) (kein Vorzeichen, immer positiv)
5. longint: 4 Bytes (1 Bit als Vorzeichen)

13.2 Gleitkommazahlen

Bei der Deklaration einer Variable vom Typ *real* wird ein Speicherplatz von 6 Bytes (48 Bit) belegt. Der Datentyp *real* wird in Form von Mantisse und Exponent dargestellt: $\pm 1.6578..45E+01$. Diese Form der Darstellung gilt für sämtliche Gleitkomma-Variablentypen, wie *real, single, double, extended*. Der Dezimalpunkt wird nicht explizit gespeichert, sondern implizit angenommen. Es ergeben sich für die unterschiedlichen Gleitkomma-Variablentypen auch unterschiedliche Speicherplatzanforderungen und damit auch unterschiedliche Zahlenbereiche:

1. single: 4 Bytes
2. real: 6 Bytes
3. double: 8 Bytes
4. extended: 10 Bytes

13.3 Aufgaben zu den Datenformaten

1. Bestimmen Sie unter Verwendung des Turbo Pascal Hilfesystems die Zahlenbereiche für jeden Gleitkomma- Variablentyp!
2. Deklarieren Sie einen Vektor mit 3000 Elementen vom Typ *real* und belegen Sie jedes Element mit einer Zufallszahl (*random*-Funktion) ! Wieviel Speicherplatz benötigen alle Elemente des Vektor zusammen (in Bytes !) ?
3. Speichern Sie die in Aufgabe 2 erzeugten Zufallszahlen in einer *typisierten* Datei! Wieviel Speicherplatz belegt diese Datei auf der Festplatte (in Bytes)?
4. Speichern Sie die in Aufgabe 2 erzeugten Zufallszahlen in einer Datei vom Typ *text* ab! Wieviel Speicherplatz belegt diese Datei auf der Festplatte (in Bytes)?

5. Wie erklärt sich der unterschiedliche Speicherplatzbedarf von typisierten und von Textdateien?

14 Dynamische Variablen

14.1 Was sind dynamische Variablen?

Wenn bisher von Variablen gesprochen wurde, so verband sich damit die Vorstellung, daß in einer Variablendeklaration ein Bezeichner festgelegt wurde, für den vom Compiler für die Laufzeit des Programms Speicherplatz reserviert wurde. Dieser Speicherbereich stand während der gesamten Abarbeitungszeit des Programms zur Verfügung (statische Variable). Der Zugriff zu dem Speicherbereich erfolgte über den Namen der Variablen.

Alternativ (und auch sinnvoll) ist es jedoch möglich, den Speicherplatz für eine Variable nur für die Programmlaufzeit anzufordern, in welcher die Variable auch aktiv verwendet wird. Damit kann der Programmierer festlegen bzw. darüber bestimmen, in welchem Programmabschnitt mit der Variable gearbeitet werden soll (und dieser Variable Speicherplatz zur Verfügung gestellt werden soll).

Diese so verwendeten Variablen werden als *dynamische* Variablen oder auch als *Pointervariablen* bezeichnet.

14.2 Die Deklaration

Die Deklaration von dynamischen Variablen in Pascal erfolgt unter Verwendung des Symbols $\wedge$. Der nachfolgende Ausschnitt aus dem Vereinbarungsteil eines Pascalprogrammes zeigt, wie die Variable namens *tiger* als dynamische Variable deklariert wird:

```
var tiger : ^integer;
    x,y   : ^real;
```

Eine *record*- Variable kann auf folgende Weise als eine dynamische Variable deklariert werden:

```
type adresse= record
                name  : string;
                alter : integer;
              end;

var selber : ^adresse;
```

14.3 Programmbeispiele

14.3.1 Ein einfaches Programmbeispiel

In diesem Programmbeispiel wird für die Variable x unter Verwendung der Funktion *new* Speicherplatz angefordert und zugewiesen. Mit der Funktion *dispose* wird der von dieser Variablen belegte Speicherplatz wieder freigegeben.

```
program dynamische_variable_1;
  var x: ^real;
```

```
begin
 new(x); { Zuweisung von Speicherplatz fuer die Variable x }
 x^:=random;
 writeln('Zufallszahl',x^);
 dispose(x);  { Freigabe des von x belegten Speicherplatzes }
end.
```

Alternativ zu *new* und *dispose* kann mit mit den Funktionen *mark* und *release* gearbeitet werden, auf die hier nicht näher eingegangen wird.

14.3.2 Noch ein Programmbeispiel

In diesem Programmbeispiel wird für die Variable *x* unter Verwendung der Funktion *new* Speicherplatz angefordert und zugewiesen. Mit der Funktion *dispose* wird der von dieser Variablen belegte Speicherplatz wieder freigegeben.

```
program dynamische_variable_2;
  var x,y,z: ^real;
begin
 new(x); { Zuweisung von Speicherplatz fuer die Variable x }
 new(y); { Zuweisung von Speicherplatz fuer die Variable y }
 x^:=10;
 y^:=20;
 writeln('x= ',x^:3:1,' y= ',y^:3:1);

 z:=x;   { Der Adresszeiger z zeigt nun auf die Adresse von x }
 writeln('z= ',z^:3:1);

 z^:=3;       { Vorher stand hier die Zahl 10 ! }
 writeln('z= ',z^:3:1);

 z:=y;   { Der Adresszeiger z zeigt nun auf die Adresse von y }
 writeln('z= ',z^:3:1,' 'y= ',y^:3:1);

 dispose(x);  { Freigabe des von x belegten Speicherplatzes }
 dispose(y);  { Freigabe des von y belegten Speicherplatzes }

 new(z); { Zuweisung von Speicherplatz fuer die Variable z }
 z^:=0.3;
 dispose(z);  { Freigabe des von y belegten Speicherplatzes }
 readln;
end.
```

Mit der Funktion *new* wird nur der Adreßzeiger *x* auf eine definierte Speicheradresse gesetzt und es werden die nachfolgenden 6 Byte für die Gleitkommazahl von Typ *real* reserviert.

14.3.3 Die Funktionen memavail und maxavail

Mit Hilfe der Funktionen *memavail* und *maxavail* kann man Informationen über den freien Speicher erhalten:

memavail Gesamtmenge des noch freien Speicherplatzes im Heap

```
writeln('Freier Speicher vor <new>: ',memavail,' Bytes');
new(x);
writeln('Freier Speicher nach <new>: ',memavail,' Bytes');
```

maxavail Umfang des größten freien Blocks im Heap

```
writeln('Groesster freier Block: ',maxavail,' Bytes');
```

14.3.4 Die Speicherorganisation

⇑Heap	*Heapzeiger, wächst in Richtung aufsteigender Adresse*
⇓Stack	*Stackzeiger, wächst in Richtung absteigender Adressen*
⇑statischer Datenbereich	*hier werden die statischen Variablen abgelegt*
⇑Objectcode	*hier werden die ausführbaren Programme abgelegt*
⇑Betriebsystem	*hier liegt das Betriebssystem*

14.4 Records als dynamische Variablen

Das nachfolgende Programm zeigt die Arbeit mit dynamischen *record*- Variablen:

```
program dynamische_record_variable;
type baum = record
              name  : string;
              alter : integer;
              hoehe : real;
            end;

     baum_dyn = ^baum; {record als dynamische Variable }

var laub_1, laub_2 : baum_dyn;

begin
 new(laub_1); { Zuweisung von Speicherplatz laub_1 }
   laub_1^.name:='Ahorn';
   laub_1^.alter:=30;
   laub_1^.hoehe:=13.37;
 dispose(laub_1);  { Freigabe des belegten Speicherplatzes }
```

```
 new(laub_2); { Zuweisung von Speicherplatz laub_1 }
   laub_2^.name:='esche';
   laub_2^.alter:=17;
   laub_2^.hoehe:=6.72;
 dispose(laub_2);  { Freigabe des belegten Speicherplatzes }
end.
```

14.5 Übungsaufgaben

Lösen Sie die folgenden Aufgaben unter Verwendung von dynamischen Variablen!

1. Was sind statische Variablen? Was sind dynamische Variablen? Was ist der Unterschied zwischen statischen und dynamischen Variablen?

2. Erklären Sie folgende Wörter:

   - Speicheradresse
   - Speicherinhalt
   - Offset

   Versuchen Sie diese Begriffe mit Hilfe von Beispielen zu erklären!

3. Geben Sie unter Verwendung der *repeat-until*-Anweisung den Umfang eines Kreises in Abhängigkeit vom Kreisradius im Bereich von 23,5 cm bis 43,5 cm auf dem Bildschirm aus! Der Kreisradius wird bei jedem Durchlauf um 0,5 cm erhöht.

4. Deklarieren Sie einen Vektor mit 3000 Elementen vom Typ real und belegen Sie jedes Element mit einer Zufallszahl (*random*-Funktion) !

5. Bestimmen Sie die Summe aller Feldelemente mit geradem Index unter verwendung der *mod*-Funktion!

6. Deklarieren Sie eine Matrix mit einer Struktur von 100 Zeilen und 30 Spalten vom Typ real und initialisieren Sie jedes Element mit dem Wert 0 ! Belegen Sie jedes Matrixelement mit einer Zufallzahl (*random*-Funktion) !

7. Berechnen Sie die Summe über alle Matrixelemente unter Verwendung der

   - *for-to-do*
   - *repeat-until*
   - *while-do*

   Anweisung ! Achten Sie auf die Feldgrenzen !

8. Berechnen Sie die Summe der Hauptdiagonalelemente !

15 Listenprogrammierung

15.1 Was sind Listen?

Eine Liste kann man sich als die Verkettung von *records* darstellen. Jedes *record*- Element enhält in seiner Struktur im Form einer Pointervariable ein zusätzliches Element, welches auf die Adresse des nächsten Listenelementes bzw. *record* zeigt.
Bei einer Listenstruktur ist es nicht mehr möglich, direkt auf den jeweiligen Adreßeintrag zuzugreifen - es muß sich sozusagen *durchgehangelt* werden. Der bei einem Array vorhandene Index ist hier nicht mehr vorhanden. Stattdessen wird das *record*-Element (also der Adreßeintrag) um ein Element erweitert: die nächste **Adresse** (auch Seitennumer in einem Buch genannt). Im nachfolgenden Programmbeispiel ist der Aufbau einer solchen *record*- Struktur dargestellt:

```
type berlin = ^adresse;
     adresse = record
                 vorname:string;
                 nachname:string;
                 naechster: berlin;
                    { Zeiger auf das naechste Element }
               end;

var a1,a2 : berlin;

begin
  new(a1);
    a1^.vorname:='Monika';
    a1^.name:='Kaiser';

  new(a2);
    a2^.vorname:='Joschka';
    a2^.name:='Fischer';

  dispose(a2);
  dispose(a1);
end.
```

Die Regel, daß in Deklarationsanweisungen (hier im *type*- Abschnitt) nur auf Datentypen Bezug genommen werden kann, die zuvor definiert wurden, läßt in Bezug auf Zeiger eine Ausnahme zu: Der Basisdatentyp (*adresse*) einer Zeigervariablen (*berlin*) darf nachfolgend definiert werden, vorausgesetzt, dies geschieht innerhalb des gleichen Blocks.

In dem Programm ist noch keine Verknüpfung von zwei Adressen erfolgt. In dem *record*- Element sind nur Informationen über *Vorname* und *Nachname* enthalten - aber noch kein Verweis auf die Adresse des nächsten Adreßeintrages. Das werden wir im nächsten Programm sehen:

15.2 Die Verknüpfung von Records zu Listen

Im nachfolgenden Programm werden drei Adreßfelder *adr1*, *adr2* und *adr3* zu einer Liste verkettet:

```
program endlich;
type berlin = ^adresse;
     adresse = record
                 vorname:string;
                 nachname:string;
                 naechster: berlin;
                       { Zeiger auf das naechste Element }
               end;

var adr1, adr2, adr3  : berlin;
    anfang, vor, aktuell : berlin;

begin
  new(adr1);    { Speicherplatzanforderung fuer adr1 }
    adr1^.vorname:='Monika';
    adr1^.name:='Kaiser';
    adr1^.naechster:=nil;  { hier wird die Speicheradresse  }
                           { des naechsten records abgelegt }

  new(adr2);    { Speicherplatzanforderung fuer adr2 }
    adr2^.vorname:='Heinrich';
    adr2^.name:='Heine';
    adr2^.naechster:=nil;
          { Speicheradresse des naechsten records !}

  { Jetzt erfolgt die Verknuepfung von adr1 und adr2 }
  adr1^.naechster:=adr2;
                  { Ob ich das jetzt verstanden habe ? }

  new(adr3);    { Speicherplatzanforderung fuer adr2 }
    adr3^.vorname:='Dalai';
    adr3^.name:='Lama';
    adr3^.naechster:=nil;
          { Speicheradresse des naechsten records! }

  { Jetzt erfolgt die Verknuepfung von adr2 und adr3 }
  adr2^.naechster:=adr3;

  { writeln(adr3); ist nicht erlaubt !! }
  { Die Ausgabe von Speicheradressen ist nicht erlaubt!}
```

```
  { geht nur mit dem Debugger }

  dispose(a3);
  dispose(a2);
  dispose(a1);
end.
```

Die in diesem Quelltext aufgeführten Variablen *anfang*, *vor* und *aktuell* (adr1, adr2, adr3) stellen genau die notwendigen Parameter zur Charakterisierung einer Liste dar:

anfang die Anfangsadresse einer Liste

vor das vorletzte Element der Liste, welches die Adresse des letzten Elementes enthält

aktuell das aktuelle bzw. letzte Element einer Liste

15.3 Allgemeine Programmierung von Listen

Das nachfolgende Quelltext stellt den schematischen Ablauf zur Programmierung einer Liste dar:

```
program endlich;
type berlin = ^adresse;
     adresse = record
                 vorname:string;
                 nachname:string;
               { Das ist der Zeiger auf das naechste Element }
                 naechster: berlin;
               end;

var  anfang, vor, aktuell : berlin;

begin
  new(aktuell); {Speicherplatzanforderung fuer Element Nr. 1}
    aktuell^.vorname:='Monika';
    aktuell^.name:='Kaiser';
    aktuell^.naechster:=nil;

  anfang:=aktuell;  { Anfang der Liste merken !! }
  vor:=aktuell;

  new(aktuell); {Speicherplatzanforderung fuer Element Nr. 2}
    aktuell^.vorname:='Joschka';
    aktuell^.name:='Fischer';
    aktuell^.naechster:=nil;
```

```
  { Adresse im vorhergehenden Element abspeichern ! }
  vor^.naechster:=aktuell;

  vor:=aktuell;   { Merken der aktuellen Adresse }

  new(aktuell); {Speicherplatzanforderung fuer Element Nr. 3}
    aktuell^.vorname:='Dalai';
    aktuell^.name:='Lama';
    aktuell^.naechster:=nil;

  { Adresse im vorhergehenden Element abspeichern ! }
  vor^.naechster:=aktuell;

  vor:=aktuell;   { Merken der aktuellen Adresse }

  new(aktuell); {Speicherplatzanforderung fuer Element Nr. 4}
    aktuell^.vorname:='Georg';
    aktuell^.name:='Bush';
    aktuell^.naechster:=nil;

  { Adresse im vorhergehenden Element abspeichern ! }
  vor^.naechster:=aktuell;

  vor:=aktuell;   { Merken der aktuellen Adresse }

end.
```

15.4 Übungsaufgaben

1. Erzeugen Sie eine kleine Adreßdatenbank unter Verwendung von *record*-Variablen mit 3 Adreßeinträgen! Jeder Adreßeintrag soll den Nachnamen und das Alter der Person enthalten. Deklarieren Sie die drei *record*- Variablen

   a) als statische Variablen!

   b) als dynamische Variablen!

2. Erweitern Sie die in der vorhergehenden Aufgabe definierte *record*- Struktur um ein Element, welches die Speicheradresse des nächsten Elementes enthalten wird!

3. Verknüpfen Sie die drei Adreßeinträge zu einer Liste!

4. Geben Sie den Inhalt aller drei Listenelemente auf dem Bildschirm aus!

5. Erweitern Sie die Liste um einen zusätzlichen Adreßeintrag am Ende der Liste!

6. Wie können Sie den Inhalt des *dritten* Listenelementes auf dem Bildschirm ausgeben?

7. Wie können Sie den Inhalt des *letzten* Listenelementes auf dem Bildschirm ausgeben?

8. Löschen Sie den zweiten Eintrag aus der Liste!

9. Speichern Sie den Anfang (die Anfangsadresse)der Liste auf einer dynamischen Variable ab!

10. Wie können Sie bestimmen aus wieviel Elementen eine Liste besteht?

11. Schreiben Sie ein Unterprogramm, welches ein Listenelement an eine vorhandene Liste anhängt!

    - Input: Datensatz, Anfangsadresse der Liste

12. Schreiben Sie ein Unterprogramm, welches ein Listenelement in einer vorhandenen Liste löscht!

    - Input: Anfangsadresse der Liste, Numer des Listenelementes

13. Durchsuchen Sie die Liste nach einem vorgegebenen Suchmuster (zum Beispiel alle Personen die älter als 20 Jahre sind)!